SYS863
Internet industriel des objets

Laboratoire 1

Lokman Sboui, Ph.D.
Professeur en génie des systèmes
École de technologie supérieure
Montréal, Québec

Hiver 2023

Table des Matières - Laboratoire 1

1. Introduction

1.1 Objectifs

Ce laboratoire de IIdO constitue une introduction à la programmation du Raspberry Pi 3 B en langage C et de l'utilisation du matériel du laboratoire :

- Utilisation du logiciel Geany pour écrire, compiler et tester et les programmes en C.
- Apprendre comment faire la configuration des les broches du GPIO,
- Programmation des processus légers (thread).

Ensuite, vous allez développer un exemple de couche de perception qui est l'asservissement de température à l'aide du Raspberry Pi qui est branché à

- un capteur de température (thermocouple du type K) ;
- une lampe de 12 Volts pour augmenter la température autour du thermocouple.

Vous devrez implanter deux approches de commandes pour faire un asservissement de température une commande de type thermostat (ON/OFF) ; et une commande de type PID. Pour réussir cet exploit, vous devrez :

- Réaliser la lecture du thermocouple via le protocole SPI implanté de façon software.
- Écrire des valeurs au PWM pour commander l'intensité d'une lampe de 12 Volts.
- Réaliser une commande de type thermostat (ON/OFF) et de type PID pour asservir la température de la lampe. La fréquence d'acquisition sera de 1 Hertz, la dynamique de ce système étant très lente (vous allez le constater).

1.2 Évaluation

Ce laboratoire représente 10% de la note finale. Un rapport et le listing final doivent être remis via le site Moodle du cours. Les livrables demandés seront identifiés avec ce symbole "↬" et seront écrit en rouge.

L'évaluation des listings est basée sur les critères suivants :

▷ Qualité et quantité des commentaires ;

▷ Toutes les fonctions que vous ajoutez doivent comporter un bloc d'entêtes avec les informations suivantes :

- ◎ Nom de la fonction ;
- ◎ Brève description de la tâche effectuée ;
- ◎ Paramètres d'entrée ;
- ◎ Paramètres de sortie ;
- ◎ Programmes ou fonctions appelant cette fonction ;
- ◎ Fonctions appelées par cette fonction ;

1.3 Format à utiliser

Le rapport doit être soumis sous forme de fichier .pdf avec le nom **EquipeX_LabY_Rapport.pdf** avec X la lettre de votre équipe et Y le numéro du laboratoire.

Pour les autres fichiers (ex :listings en C) le nom doit être : **EquipeX_LabY_PartieZ.c** avec Z le numéro de la partie dans la description du laboratoire reliée au listing.

Le rapport de laboratoire doit se conformer le plus possible au format suivant :

- ▷ Page de présentation ;
- ▷ Buts et objectifs du laboratoire ;
- ▷ Expliquer votre démarche de conception ;
- ▷ Expliquer ce que vous observez ;
- ▷ Pour chaque programme que l'on vous demande d'annexer au rapport de laboratoire :
  - ◎ Explication du fonctionnement de chaque programme (ce peut être via un ordinogramme) ;
  - ◎ " Listing " des fichiers sources (les sections ou fonctions modifiées ou ajoutées) ;
- ▷ Problématiques rencontrées ;
- ▷ Conclusion.

1.4 Préparatifs

Pour vous préparer au laboratoire, il peut être nécessaire de consulter quelques documents ainsi que lire le document - Introduction aux Laboratoires - .

- ▷ `https://www.tutorialspoint.com/cprogramming/cprogramming_tutorial.pdf` : Tutoriel sur le langage C ;
- ▷ `https://www.raspberrypi.org/magpi-issues/Essentials_C_v1.pdf` : Un autre référence sur le C avec le Raspberry Pi ;
- ▷ `http://www.circuitbasics.com/useful-raspberry-pi-commands/` : Commandes dans l'environnement fenêtre de commande ;
- • `https://www.sparkfun.com/datasheets/BreakoutBoards/MCP4725.pdf` : Fiche technique du convertisseur numérique analogique ;
- • `https://cdn-shop.adafruit.com/datasheets/ads1015.pdf` : Fiche technique du convertisseur analogique numérique ;
- ▷ `http://homes.dsi.unimi.it/~boccignone/GiuseppeBoccignone_webpage/MatDidatSOD2009_files/pthread-Tutorial.pdf` : Texte sur les "thread".

Assurez vous d'avoir les deux fiches techniques (identifiées dans la liste ci-haut par •) sous la main pour le laboratoire.

2. Travaux du Laboratoire 1

2.1 Partie 1 - Initiation à la programmation pour l'IIoT

2.1.1 Compiler et tester un premier programme en C

Le premier programme en C que vous avez à faire fera clignoter un DEL à une fréquence de 1 Hertz. Le Listing 2.1 est le programme que vous aurez à entrer sur le Raspberry Pi. Cela vous permettra de vous familiariser avec l'éditeur Geany.

Listing 2.1 – Clignotement d'un DEL à 1 Hz

```
1  /*
2   * Cligne_1_Hz.c
3   *
4   * Ce programme fait clignoter un DEL à une fréquence de 1 Hz.
5   * L'arrêt sera provoqué par un appui sur le bouton-poussoir 1.
6   * Aucun paramètre en entrée.
7   *
8   */
9
10 #include <stdio.h>
11 #include <bcm2835.h>
12
13 int main(int argc, char **argv)
14 {
15         // Initialisation du bcm2835
16         if (!bcm2835_init()){
17                 return 1;
18         }
19
20         // Configuration du GPIO pour DEL 1 (rouge)
21         bcm2835_gpio_fsel(20, BCM2835_GPIO_FSEL_OUTP);
22         // Configuration du GPIO pour bouton-poussoir 1
23         bcm2835_gpio_fsel(19, BCM2835_GPIO_FSEL_INPT);
24
```

```
25          // Tant que bouton-poussoir #1 non actionné, clignoter
26          // DEL allumée 500 ms et éteinte 500 ms
27          while(bcm2835_gpio_lev(19)){
28                  bcm2835_gpio_write(20, HIGH);
29                  bcm2835_delay(500);
30                  bcm2835_gpio_write(20, LOW);
31                  bcm2835_delay(500);
32          }
33
34          // Libérer les broches du GPIO
35          bcm2835_close();
36          return 0;
37  }
```

La ligne 11 permet d'inclure la librairie du bcm2835 qui contient toutes des déclarations de fonctions et de constantes nécessaires pour le fonctionnement du programme. Vous devrez l'inclure dans tous les programmes utilisant les GPIO du Raspberry Pi.

La ligne 16 sert à lancer l'initialisation du bcm2835, ce qui rendra possible l'utilisation des broches du GPIO. Si l'initialisation ne se fait pas correctement, le programme s'arrête avec un statut égal à "1" (à cause du `return 1;`), indiquant qu'il y a eu un problème.

Si tout se passe bien, on arrive aux lignes 21 et 23 où l'on associe la broche #20 du GPIO à une sortie, puisque le DEL #1 (rouge) y est branché et la broche #19 du GPIO à une entrée, car elle est branchée au bouton-poussoir #1 (noir).

La boucle `while`, de la ligne 27 à la ligne 32, est l'endroit ou le clignotement du DEL se produit. Le DEL s'allume avec l'instruction `bcm2835_gpio_write(20, HIGH)` et s'éteint avec `bcm2835_gpio_write(20, LOW)`. La fréquence de 1 Hertz est obtenue via deux délais de même durée, soit 500 ms : `bcm2835_delay(500)`.

On quitte la boucle `while` lorsque le bouton-poussoir noir est enfoncé. Cela est vérifié par l'instruction `bcm2835_gpio_lev(19)`. Notez que le bouton-poussoir doit être enfoncé lorsque l'instruction `while` est en cours d'exécution, sinon la boucle reprend. Cette façon de vérifier l'état du bouton est appelée le *polling*. Cette approche est facile à implanter mais comporte cet inconvénient. Si la fréquence du clignotement était de 100 Hertz, ce ne serait pas trop ennuyeux, mais ici, il faut appuyer lorsque le DEL est éteint, donc au moins pendant 1/2 seconde.

TABLE 2.1 – Liste des broches du GPIO utilisées sur le Raspberry Pi

Broche	GPIO	Fonction
3	2	"Convertisseurs AN/NA" - Signal SDA du I2C
5	3	"Convertisseurs AN/NA" - Signal SCL du I2C
12	18	"Lampe de 12 Volts" - Sortie logique ou PWM (canal 0)
16	23	"Software SPI avec le MAX31855" - signal MISO
18	24	"Software SPI avec le MAX31855" - signal CS (sélection de l'esclave)
22	25	"Software SPI avec le MAX31855" - signal SCK
35	19	Bouton poussoir noir (#1)
37	26	Bouton poussoir rouge (#2)
38	20	DEL rouge (#1)
40	21	DEL jaune (#2)

Instructions :

1. Écrire et exécuter le programme du Listing 2.1. Testez son bon fonctionnement.
2. Ajouter le second DEL (orange) et le faire clignoter en opposition avec le DEL rouge (allumé quand l'autre est éteint et vice-versa). S'assurer que les deux DEL soient éteints lorsque l'on quitte le programme.
3. Ajouter le second bouton dans la condition du `while` pour imposer que les deux boutons soient enfoncés pour stopper le programme.
4. Ajuster la fréquence du clignotement des deux DEL à 0.1 Hertz.
5. ↬ Quel problème observez vous lorsque vous testez le programme ?
6. **Montrez le fonctionnement au professeur.**

Note :

Le listing obtenu à la fin de cette manipulation n'est pas à remettre.

2.1.2 Compiler et tester un programme utilisant un thread

Dans cette seconde partie, nous ferons clignoter nos DEL, mais via un thread qui s'exécutera en parallèle avec le `main`. Le Listing 2.2 montre ce programme ou l'on fait clignoter le DEL #1 (rouge) à 1 Hertz.

Listing 2.2 – Clignotement d'un DEL à 1 Hz avec un thread

```
1  /*
2   * Thread_Cligne_1_Hz.c
3   *
4   * Ce programme fait clignoter un DEL à une fréquence de 1 Hz.
5   * Le clignotement se fait à l'intérieur d'un thread.
6   * L'arrêt sera provoqué par un appui sur le bouton-poussoir 1.
7   * Aucun paramètre en entrée.
8   *
9   */
10
11 #include <unistd.h>
12 #include <stdio.h>
13 #include <bcm2835.h>
14 #include <pthread.h>
15
16 /* Fonction clignote qui sera appelée par le thread "cligne"
17  *
18  * Elle fait clignoter le DEL rouge à 1 Hertz.
19  * La fonction ne retourne aucune valeur.
20  * La fonction n'exige aucun paramètre en entrée.
21  */
22 void *clignote()
23 {
24         // Boucle infinie pour le clignotement du DEL
25         while(1){
26                 bcm2835_gpio_write(20, HIGH);
27                 bcm2835_delay(500);
28                 bcm2835_gpio_write(20, LOW);
29                 bcm2835_delay(500);
30         }
31         pthread_exit(NULL);
```

```
32 }
33
34 /* Fonction main
35  *
36  * Elle configure le GPIO et le thread.
37  *
38  * La fonction ne retourne aucune valeur.
39  * La fonction n'exige aucun paramètre en entrée.
40  */
41 int main(int argc, char **argv)
42 {
43         // Identificateur du thread
44         pthread_t cligne;
45
46         // Initialisation du bcm2835
47         if (!bcm2835_init()){
48                 return 1;
49         }
50
51         // Configuration du GPIO pour DEL 1 (rouge)
52         bcm2835_gpio_fsel(20, BCM2835_GPIO_FSEL_OUTP);
53         // Configuration du GPIO pour bouton-poussoir 1
54         bcm2835_gpio_fsel(19, BCM2835_GPIO_FSEL_INPT);
55
56         // Création du thread "cligne".
57         // Lien avec la fonction clignote.
58         // Cette dernière n'exige pas de paramètres.
59         pthread_create(&cligne, NULL, &clignote, NULL);
60
61         // Boucle tant que le bouton-poussoir est non enfoncé
62         while(bcm2835_gpio_lev(19)){
63                 usleep(1000); // Délai de 1 ms !!!
64         }
65
66         // Si bouton-poussoir enfoncé, arrêt immédiat du thread
67     pthread_cancel(cligne);
68     // Attente de l'arrêt du thread
69     pthread_join(cligne, NULL);
70
71     // Éteindre le DEL rouge
72     bcm2835_gpio_write(20, LOW);
73     // Libérer le GPIO
74     bcm2835_close();
75     return 0;
76 }
```

On constate de nombreuses ressemblances avec le Listing 2.1. La boucle qui faisait clignoter la DEL est maintenant dans la fonction `clignote`, associée au thread `cligne`. Le bouton-poussoir est la condition de sortie de la boucle `while` dans le programme principal. Cette condition est vérifiée à toutes les millisecondes. Donc, on aura une réaction immédiate lors de l'appui sur le bouton, et ce sera indépendant de la fréquence de clignotement choisie.

La nouveauté est l'utilisation de fonctions associées au thread, ce qui nécessite une référence à la librairie des threads POSIX (ou pthreads) via le `#include <pthread.h>`.

En ligne 45, on déclare la structure `pthread_t cligne`. Cette structure de donnée de type `pthread_t` est un peu opaque et difficile à comprendre. Toutefois, il suffit de savoir qu'une zone mémoire est

définie pour stocker les paramètres importants pour le thread identifié par `cligne`.

Parmi ces paramètres, il y a l'adresse de la fonction associée au thread `cligne` et l'adresse où seront situés les paramètres de cette fonction (s'il y en a). Ces paramètres sont fournis par la fonction `pthread_create(&cligne, NULL, &clignote, NULL)` en ligne 60. On retrouve l'identificateur du thread, puis un paramètre rarement utilisé et laissé à `NULL` pour indiquer que l'on utilise la configuration par défaut. Vient ensuite l'adresse de la fonction et les paramètres (quand il n'y en a pas, comme ici, on inscrit `NULL`). Il faut bien sûr définir la fonction qui sera associée au thread.

La fonction `pthread_create` créé et démarre le thread. Le thread peut s'arrêter parce que la fonction associée se termine, ce qui ne sera pas le cas ici en raison de la boucle infinie. Il peut s'arrêter si on demande son annulation, ce qui est fait en ligne 68 avec la fonction `pthread_cancel (cligne)`. Cette ligne ne sera exécutée que si l'on sort de la boucle `while` qui la précède, i.e., que lorsque l'on appui sur le bouton-poussoir #1.

Pour s'assurer que le thread soit bel et bien terminé avant le programme principal, on utilise la fonction `pthread_join`. On indique l'identifiant du thread et l'adresse des valeurs que la fonction nous retourne. Ici, comme la fonction ne fait que clignoter la DEL et ne retourne rien alors on retrouve en ligne 70 : `pthread_join(cligne, NULL)`, avec le `NULL` comme second paramètre pour indiquer que l'on attend pas de valeurs de retour.

Instructions :

1. Écrire et exécuter le programme du Listing 2.2. Testez son bon fonctionnement.
2. Ajouter le second DEL (orange) et le faire clignoter en opposition avec le DEL rouge (allumé quand l'autre est éteint et vice-versa), **mais dans un second thread**. S'assurer que les deux DEL soient éteints lorsque l'on quitte le programme.
3. Ajouter le second bouton dans la condition du `while` du programme principal pour imposer que les deux boutons soient enfoncés pour stopper le programme.
4. Ajuster la fréquence du clignotement des deux DEL à 0.1 Hertz (période de 10 secondes).
5. ↬ Est-ce que le temps de réponse pour stopper le programme s'est amélioré ? Expliquez votre réponse.

Note :

Le listing obtenu à la fin de cette manipulation n'est pas à remettre.

2.1.3 Compiler et tester un programme avec délai précis

Pour de basses fréquences, l'utilisation de la fonction `bcm2835_delay(int delai)` peut donner des délais relativement précis mais pour des fréquences plus élevées, on peut perdre de la précision car le microcontrôleur fait plein de choses en plus de s'occuper de l'exécution de notre programme.

On contourne ce problème en utilisant une des fonctions de la bibliothèque `time.h`. La fonction `unsigned long clock()` retourne le nombre de coups d'horloge qui se sont produits depuis le lancement du programme. Un des paramètres contenu dans cette librairie est le nombre de coups d'horloge du CPU par secondes (`CLOCKS_PER_SEC`).

Si nous avons une certaine section de code dans le thread qui doit être exécuté à une certaine cadence précise, on peut comptabiliser le nombre de coups d'horloge avant cette section, puis après. Ce nombre divisé par le nombre de coups de d'horloge par seconde permet d'obtenir le temps d'exécution de la section de code en secondes.

Si on désire exécuter cette section de code à tous les δt secondes, il suffit de faire un délai du nombre de secondes nécessaires pour obtenir la cadence voulue. Cette opération peut sembler

inutile pour le clignotement d'un DEL, mais devient très intéressante lorsque l'on échantillonne des signaux à des fréquences relativement élevées (dans notre cas 1000 Hz).

Le Listing 2.3 montre un programme de clignotement de DEL dont la fréquence désirée est de 10 Hz. Elle ressemble énormément au Listing 2.2 excepté dans la fonction `void *clignote()`.

Listing 2.3 – Clignotement de 10 Hz avec utilisation de fonctions de la librairie time.h

```
/*
 * Thread_Cligne_10_Hz_time.c
 *
 * Ce programme fait clignoter un DEL à une fréquence de 10 Hz.
 * Le clignotement se fait à l'intérieur d'un thread et le délai est
 * ajusté avec des fonctions de la librarie "time.h".
 * L'arrêt sera provoqué par un appui sur le bouton-poussoir 1.
 * Aucun paramètre en entrée.
 *
 */

#include <unistd.h>
#include <stdio.h>
#include <bcm2835.h>
#include <pthread.h>
#include <time.h>

double frequence = 10.0; // Hertz : fréquence de clignotement désirée

/* Fonction clignote qui sera appelée par le thread "cligne"
 *
 * Elle fait clignoter le DEL rouge à 1 Hertz.
 *
 * La fonction ne retourne aucune valeur.
 * La fonction n'exige aucun paramètre en entrée.
 */
void *clignote()
{
        clock_t debut, fin; // Variables de temps
        double demiPer = 1/(2.0*frequence); // Calcul de la demi-période
        int etat = 0; // État du DEL

        debut = clock(); // Temps écoulé depuis le lancement du programme

        // Boucle infinie pour le clignotement du DEL
        while(1){
                if (etat==0){
                        bcm2835_gpio_write(20, HIGH);
                        etat = 1;
                }
                else{
                        bcm2835_gpio_write(20, LOW);
                        etat = 0;
                }

                fin = clock(); // Temps écoulé depuis le lancement du
                    programme

                // La différence (fin-debut) donne le temps d'exécution du
```

```
                    code
49                  // On cherche une durée égale à demiPer. On compense avec un
                        délai
50                  // CLOCK_PER_SEC est le nombre de coups d'horloges en 1
                        seconde
51
52                  usleep(1000000*(demiPer-((double) (fin-debut)/((double)
                        CLOCKS_PER_SEC))));
53                  debut = fin;
54          }
55          pthread_exit(NULL);
56  }
57
58  /* Fonction main
59   *
60   * Elle configure le GPIO et le thread.
61   *
62   * La fonction ne retourne aucune valeur.
63   * La fonction n'exige aucun paramètre en entrée.
64   */
65  int main(int argc, char **argv)
66  {
67          // Identificateur du thread
68          pthread_t cligne;
69
70          // Initialisation du bcm2835
71          if (!bcm2835_init()){
72                  return 1;
73          }
74
75          // Configuration du GPIO pour DEL 1 (rouge)
76          bcm2835_gpio_fsel(20, BCM2835_GPIO_FSEL_OUTP);
77          // Configuration du GPIO pour bouton-poussoir 1
78          bcm2835_gpio_fsel(19, BCM2835_GPIO_FSEL_INPT);
79
80          // Création du thread "cligne".
81          // Lien avec la fonction clignote.
82          // Cette dernière n'exige pas de paramètres.
83          pthread_create(&cligne, NULL, &clignote, NULL);
84
85          // Boucle tant que le bouton-poussoir est non enfoncé
86          while(bcm2835_gpio_lev(19)){
87                  usleep(1000); // Délai de 1 ms !!!
88          }
89
90          // Si bouton-poussoir enfoncé, arrêt immédiat du thread
91      pthread_cancel(cligne);
92      // Attente de l'arrêt du thread
93      pthread_join(cligne, NULL);
94
95      // Éteindre le DEL rouge
96      bcm2835_gpio_write(20, LOW);
97      // Libérer le GPIO
98      bcm2835_close();
99      return 0;
100 }
```

Une variable globale `frequence` est définie en ligne 18. Elle contient la fréquence désirée de clignotement. Comme le clignotement d'une DEL comporte deux phases : allumée et éteinte, il faut calculer la durée de chaque phase qui est simplement la moitié de la période du clignotement. Cette valeur est calculée lors de la déclaration de la variable `demiPer` en ligne 30. La phase ou l'on est est représentée par la variable `etat` définie en ligne 31.

L'inclusion de la librairie `time.h` en ligne 16 permet de déclarer les variables `debut` et `fin` en ligne 29. Ce sont des variables de type `clock_t` qui est ni plus ni moins qu'un entier long non signé représentant le nombre de coups d'horloges donnés depuis le lancement de l'exécution du programme.

En ligne 33, on mesure le nombre de coups d'horloge enregistrés juste avant la boucle `while`. En ligne 46, on mesure le nombre de coups d'horloges juste après l'exécution du corps de la routine `void *clignote()`. La différence entre ces deux valeurs donne le nombre de coups d'horloge qui se sont donnés entre ces deux lignes de code. Pour compenser le fait que cette durée est inférieurs à la durée de la demi période, on calcule de délai à appliquer en ligne 52. En cas de variation de la durée entre `debut` et `fin`, ce délai sera ajusté en conséquence. Enfin, on charge la valeur de `fin` dans `debut` en ligne 53.

Instructions :

1. Écrire et exécuter le programme du Listing 2.3. Testez son bon fonctionnement.
2. ↬ Le listing obtenu à la fin de cette manipulation doit être annexé à votre rapport.
3. Garder ce programme, car vous allez le recycler dans le reste du laboratoire pour y insérer, **dans la section de code entre les lignes 37 et 46**, un traitement de signal (PID) qui exige d'avoir une durée précise [a].

a. Mais comme le Raspbian du Raspberry Pi n'est pas un OS pour le temps réel, cette précision est plutôt relative...

2.2 Partie 2 - Lecture de la température sur le MAX31855

2.2.1 L'utilisation du SPI sur le MAX31855

Le MAX31855 est un amplificateur de thermocouple qui sert à l'acquisition du signal venant d'un thermocouple et envoyer une valeur numérique au microcontrôleur en utilisant le protocole SPI, voir Figure 2.1.

Le protocole SPI (Serial Peripherical Interface) permet de communiquer en full-duplex avec divers périphériques, selon une approche maitre/esclave, le Raspberry Pi étant le maitre. L'esclave sera l'amplificateur de thermocouple MAX31855 et la communication ne sera pas par le SPI hardware, mais sera reproduit de façon logicielle (vous allez programmer les transactions SPI manuellement).

Le MAX31855 permet de récupérer la température mesurée par le thermocouple. La transmission se fait par le protocole SPI. La Figure 2.2 [1] montre le chronogramme des signaux échangés durant une transaction SPI.

Cette transaction implique

- la descente du signal de sélection de l'esclave (CS Chip Select), puis,
- l'envoie de 32 signaux d'horloge (SCK).

1. Toutes les figures et tables concernant le MAX31855 sont tirés de sa fiche technique.

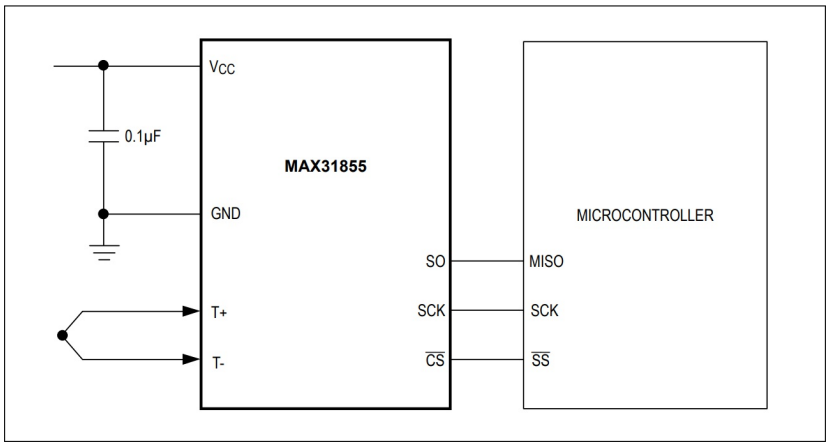

FIGURE 2.1 – Utilisation de l'amplificateur de thermocouple MAX31855.

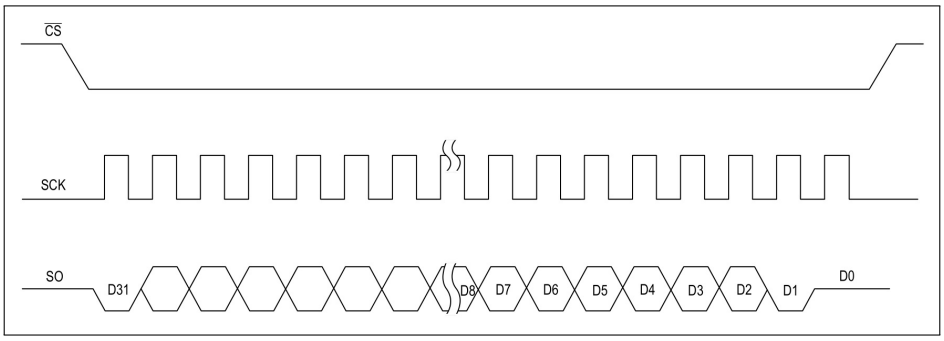

FIGURE 2.2 – Chronogramme de la communication SPI avec le MAX31855

- La transaction se termine en remontant le signal CS.

Pendant les 32 coups d'horloge, l'esclave envoie les 32 bits d'information à transmettre (S0). La donnée est présentée par l'esclave lors du front montant de l'horloge et le maître peut la capturer au front descendant.

Comme il faut imiter ce fonctionnement de façon logicielle, on devra donc changer les états logiques des signaux CS et SCK conformément au chronogramme de la Figure 2.3 et lire l'état logique du signal S0 pour obtenir l'information. On réussira cela en jouant sur les états logiques des signaux et en introduisant des délais avec la fonction `bcm2835_delayMicroseconds(int delai en microsecondes)`.

Selon sa fiche technique, le MAX31855 peut fonctionner à une vitesse de 5 MHz pour le signal SCK. La durée de l'état haut et de l'état bas de l'horloge doit excéder 100 ns (nanosecondes), ce qui correspond à la fréquence de 5 MHz.

Comme la fréquence d'acquisition du contrôleur de température est de 1 Hz (période de 1 seconde), on peut travailler à des fréquences bien inférieures à 5 MHz sans poser de problèmes.

2.2.2 La lecture du signal reçu du MAX31855

Le signal reçu de l'esclave, par le Raspberry Pi, commence par le bit le plus significatif (bit 31) et se termine par le bit le moins significatif (bit 0). La signification du bloc de 32 bits reçus est

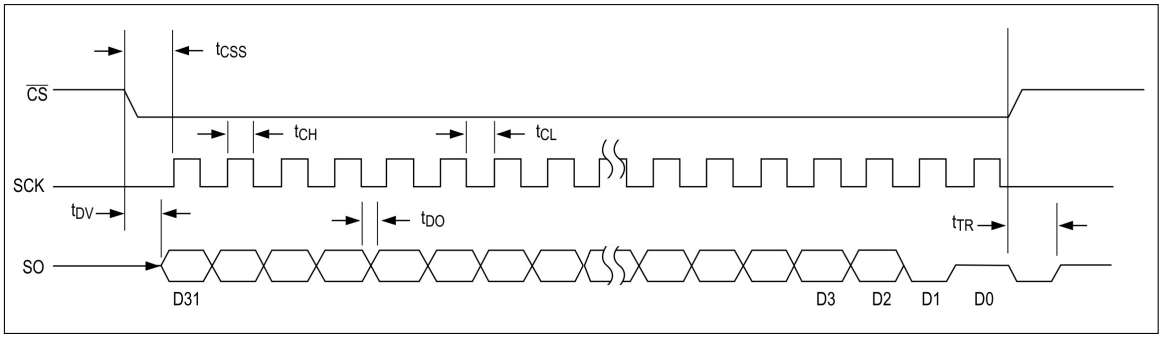

FIGURE 2.3 – Chronogramme détaillé d'une transaction SPI.

montrée dans le tableau montré en Figure 2.4.

BIT	NAME	DESCRIPTION
D[31:18]	14-Bit Thermocouple Temperature Data	These bits contain the signed 14-bit thermocouple temperature value. See Table 4.
D17	Reserved	This bit always reads 0.
D16	Fault	This bit reads at 1 when any of the SCV, SCG, or OC faults are active. Default value is 0.
D[15:4]	12-Bit Internal Temperature Data	These bits contain the signed 12-bit value of the reference junction temperature. See Table 5.
D3	Reserved	This bit always reads 0.
D2	SCV Fault	This bit is a 1 when the thermocouple is short-circuited to V_{CC}. Default value is 0.
D1	SCG Fault	This bit is a 1 when the thermocouple is short-circuited to GND. Default value is 0.
D0	OC Fault	This bit is a 1 when the thermocouple is open (no connections). Default value is 0.

FIGURE 2.4 – Signification des 32 bits du message

- Les 14 premiers bits du message (bit D31 à bit D18) contiennent la température mesurée par le thermocouple (valeur signée - en complément à 2).
- Le bit D17 est réservé et est toujours un 0.
- Le bit D16 sert à indiquer si un problème est survenu avec le thermocouple. Les trois problèmes possibles sont : 1) thermocouple court-circuité à Vcc (signalé par le bit D2) ; 2) thermocouple court-circuité à la masse (signalé par le bit D1) ; et 3) thermocouple en circuit ouvert (signalé par le bit D0). Le bit D16 indique la présence d'un problème et les bits 0, 1 et 2 précisent la nature de ce problème.
- Les 12 bits correspondant à la température interne du MAX31855 (valeur signée) sont envoyés du bit D15 au bit D4.
- Le bit D3 est réservé et est toujours 0.

Le tableau montré en Figure 2.5 résume le rôle de chaque bit.

Le format de la température du thermocouple, qui est sur 14 bits, comprend :

- Bit de signe en D31 ;
- Bit de la partie entière sur 11 bits de D30 à D20 ;
- Bit de la partie décimale sur 2 bits de D19 à D18.

La température est donc indiquée par un pas de 0.25 °C (donc la résolution du MAX31855 est de 0.25°C). Le tableau ci-dessous montre ce qu'il en est :

Le format de la température du MAX31855, qui est sur 12 bits, comprend :

	14-BIT THERMOCOUPLE TEMPERATURE DATA				RES	FAULT BIT	12-BIT INTERNAL TEMPERATURE DATA				RES	SCV BIT	SCG BIT	OC BIT
BIT	D31	D30	…	D18	D17	D16	D15	D14	…	D4	D3	D2	D1	D0
VALUE	Sign	MSB 2^{10} (1024°C)	…	LSB 2^{-2} (0.25°C)	Reserved	1 = Fault	Sign	MSB 2^{6} (64°C)	…	LSB 2^{-4} (0.0625°C)	Reserved	1 = Short to V_{CC}	1 = Short to GND	1 = Open Circuit

FIGURE 2.5 – Rôle de chacun des 32 bits

TEMPERATURE (°C)	DIGITAL OUTPUT (D[31:18])
+1600.00	0110 0100 0000 00
+1000.00	0011 1110 1000 00
+100.75	0000 0110 0100 11
+25.00	0000 0001 1001 00
0.00	0000 0000 0000 00
-0.25	1111 1111 1111 11
-1.00	1111 1111 1111 00
-250.00	1111 0000 0110 00

FIGURE 2.6 – Exemples de valeurs de températures de thermocouple

- Bit de signe en D15 ;
- Bit de la partie entière sur 7 bits de D14 à D8 ;
- Bit de la partie décimale sur 4 bits de D7 à D4.

La température est donc indiquée par pas de 0.0625 °C. Le tableau en Figure 2.7 montre quelques exemples de valeurs.

Notez que le faible pas (ou la bonne résolution du capteur) en température n'implique malheureusement pas que la mesure ait ce niveau de précision. En fait, un thermocouple de type K est précis à ± 2 °C dans la plage de température que nous utiliserons (de température de la pièce jusqu'à environ 90 °C). Le capteur mesurant la température du MAX31855 est précis à ± 2 °C aussi dans les environs de la température de la pièce.

2.2.3 Lecture et affichage de la température

Instructions :

1. Concevoir, en se basant sur Figure 2.3 et Table 2.1, la routine qui lit la température du thermocouple. La routine est déclarée comme suit : `double lecture_SPI(void)`. Elle retournera une valeur de type `double` correspondant à la température du thermocouple en °C. Vous allez devoir reproduire les signaux CS, SCLK avec le Raspberry Pi. Vous n'aurez pas à transmettre des données mais simplement lire les 32 bits du MAX31855.
   Suivre ce format, car votre routine sera intégrée dans un programme contenant une interface plus tard.

TEMPERATURE (°C)	DIGITAL OUTPUT (D[15:4])
+127.0000	0111 1111 0000
+100.5625	0110 0100 1001
+25.0000	0001 1001 0000
0.0000	0000 0000 0000
-0.0625	1111 1111 1111
-1.0000	1111 1111 0000
-20.0000	1110 1100 0000
-55.0000	1100 1001 0000

FIGURE 2.7 – Exemples de valeurs de températures du MAX31855

2. Tester votre routine dans le petit programme ou vous faisiez clignoter un LED comme la partie précedente. Vous ferez des modifications pour afficher la température à toute les secondes (en plus de clignoter le LED).
3. **Montrez le fonctionnement au professeur.**
4. Conservez cette routine pour un usage ultérieur.

Note :
Le listing obtenu à la fin de cette manipulation n'est pas à remettre, puisque la routine sera dans le listing final à remettre.

2.3 Partie 3 - Commander une lampe par PWM

2.3.1 L'utilisation du PWM

La modulation de largeur d'impulsions (Pulse Width Modulation – PWM) est une technique de modulation souvent utilisée pour commander l'amplitude moyenne d'une tension dans un moteur électrique (variateur électronique de vitesse). Dans notre cas, nous allons commander l'intensité lumineuse dans une lampe de 12 volts pour qu'elle soit plus ou moins chaude. La température de la lampe est mesurée par le thermocouple dont nous venons de faire la configuration de la lecture via le SPI.

La sortie PWM #0 sera utilisée pour commander l'intensité de la lampe de 12 volts. Cette sortie est à la broche identifiée GPIO 18. La fréquence maximale de la carte MD10C est de 20 kHz. Cette carte MD10Cde de Cytron Technologies est une carte de commande de moteurs DC d'une capacité de 10 Ampères. Il faudra donc ajuster les paramètres du PWM pour rester sous cette fréquence.

La configuration de la broche du GPIO pour l'associer au PWM est réalisée via les instructions du Listing 2.4. Cette initialisation a été abordée dans une section sur la librairie du BCM2835 dans l'introduction aux laboratoires, vous pourrez vous y référer pour plus de détails.

Listing 2.4 – Configuration du PWM

```
bcm2835_gpio_fsel(18, BCM2835_GPIO_FSEL_ALT5); // Broche 18 à PWM

bcm2835_pwm_set_clock( diviseur ); // Ajustement fréq. de base
bcm2835_pwm_set_mode(0,1,1);       // Mode du PWM
bcm2835_pwm_set_range(0, range);    // Ajustement de la plage
bcm2835_pwm_set_data(0,0);         // PWM avec duty cycle de 0%
```

Les variables entières `diviseur` et le `range` permettent d'ajuster la fréquence du PWM qui doit être sous les 20 kHz. Le `range` représentera la plage de la commande générée par le contrôleur PID. Il faudra limiter la commande entre 0 et la valeur `range`. S'assurer que la plage de `range` soit suffisamment large pour avoir une bonne résolution de commande.

Lors de l'exécution du programme, on change le duty cycle avec `bcm2835_pwm_set_data(0, commande);` avec le paramètre `commande` ajusté à la valeur correspondant au duty cycle désiré.

Lors de l'arrêt du programme, il faut s'assurer que la commande soit nulle et on devrait retrouver l'instruction `bcm2835_pwm_set_data(0,0);` avant de sortir du programme. Dans notre laboratoire, le risque pour la santé/sécurité est faible, car si on omet de faire cette commande, la lampe sera potentiellement allumée avec une intensité plus ou moins grande. Dans un contexte réel, il se pourrait que l'actionneur recevant ce signal présente des risques si le PWM continue d'agir après l'arrêt du programme.

Interrupteur à bascule "H-Bridge"

FIGURE 2.8 – Interrupteur à bascule "H-Bridge"

Cet interrupteur permet d'éviter de laisser la lampe de 12 Volts allumée et ainsi chauffer le thermocouple, ce dernier étant relativement lent à se refroidir. En position basse, la lampe est éteinte. En position haute, la lampe reçoit le signal de la sortie PWM.

2.3.2 Commande de la lampe

Instructions :

1. Concevoir la routine qui change l'intensité de la lampe de 12 Volts entre 0% et 100% puis de 100% vers 0% avec un pas de 1% un délai de 10ms entre deux changements d'intensité. Donc vous avez à changer la valeur entre 0 et la valeur maximale celle du "range" définit lors de la configuration du PWM.
2. Tester votre routine dans le petit programme ou vous faisiez clignoter un LED au début

de ce laboratoire. Vous ferez des modifications changer l'intensité de la lampe de 12 Volts (en plus de clignoter le LED).
3. **Montrez le fonctionnement au professeur.**
4. Conservez cette routine pour un usage ultérieur.

Note :
Le listing obtenu à la fin de cette manipulation n'est pas à remettre.

2.4 Partie 4 - La commande du thermostat

2.4.1 La commande de type thermostat (Tout ou rien - ON/OFF)

La première commande que l'on implantera est la commande de type thermostat (ON/OFF). Dans cette commande, on fournit au système une consigne en température $R[k]$. Si la température mesurée $y[k]$ est inférieure à la consigne $R[k]$, on ajuste le "duty cycle" du PWM à 100 % (valeur du duty cycle = range), sinon on l'ajuste à 0 % (valeur du duty cycle = 0).

Instructions :
1. Concevoir la routine qui agit comme une commande de type thermostat. La routine est déclarée comme suit : `int commande_Thermo(double consigne, double mesure)`. Elle exige deux paramètres de type `double`, la consigne de température et la mesure faite par le thermocouple. Elle retourne la commande sous la forme d'un entier dont la valeur minimale est 0 et la valeur maximale celle du "range" définit lors de la configuration du PWM. Suivre ce format, car votre routine sera intégrée dans un programme contenant une interface plus tard.
2. Tester la routine avec celle qui lit la valeur du thermocouple dans le petit programme ou vous faisiez clignoter un LED. Afficher la température et la commande à toute les secondes.
3. **Montrez le fonctionnement au professeur.**
4. Conservez cette routine pour un usage ultérieur.

Note :
Le listing obtenu à la fin de cette manipulation n'est pas à remettre.

2.4.2 La commande de type PID

La commande la plus utilisée en industrie est la commande PID. Sa simplicité d'utilisation explique sa grande popularité. La consigne en température $R[k]$ est comparée avec la température mesurée $y[k]$, la différence entre les deux valeurs étant l'erreur $e[k] = R[k] - y[k]$.

Le contrôle PID aura la forme suivante (dite forme vitesse) :

$$\begin{aligned} u[k+1] = {} & K_P(e[k] - e[k-1]) \\ & + \frac{K_I T}{2}(e[k] + e[k-1]) \\ & + \frac{K_D}{2T}(e[k] - 2e[k-1] + e[k-2]) \\ & + u[k] \end{aligned}$$

Dans cette forme, on utilise l'approximation trapézoïdale pour la partie intégrale. K_P représente le gain proportionnel, K_I représente le gain intégral et K_D représente le gain dérivé. T représente la période d'échantillonnage (dans ce laboratoire, $T = 1$ s).

La commande sera ensuite saturée avant d'être envoyée au PWM. Ainsi,

- si $u[k+1] < 0$, on force $u[k+1] = 0$;
- si $u[k+1] > u_{max}$, on force $u[k+1] = u_{max}$.

La variable u_{max} correspond à la valeur définie pour le paramètre `range` du PWM.

Instructions :

1. Concevoir la routine qui agit comme une commande de type PID de forme vitesse. La routine est déclarée comme suit : `int commande_PID(double consigne, double mesure, double gainKp, double gainKi, double gainKd)`. Elle exige cinq paramètres de type `double`, la consigne de température, la mesure faite par le thermocouple et les trois gains du PID : proportionnel, intégral et dérivé. Elle retourne la commande sous la forme d'un entier dont la valeur minimale est 0 et la valeur maximale celle du "range" définit lors de la configuration du PWM. Suivre ce format, car votre routine sera intégrée dans un programme contenant une interface plus tard.
2. Tester la routine avec celle qui lit la valeur du thermocouple dans le petit programme ou vous faisiez clignoter un LED. Afficher la température et la commande à toute les secondes.
3. **Montrez le fonctionnement au professeur.**
4. ↬ Le listing obtenu à la fin de cette manipulation doit être annexé à votre rapport.
5. ↬ Remettre un rapport expliquant la démarche que vous avez suivi pour concevoir les divers programmes demandés.

- ↬ *Optionnel - Bonus 5pts :* J'aimerai avoir votre opinion. Indiquez-moi ce que vous pensez de ce laboratoire. Quels sont les points à améliorer ? Le cours étant en construction et en processus d'amélioration continue, votre opinion m'aidera sur ce point.